```
function analyse_errors_bins(pos_estimated,score,pos, endbulges)
%analyse_errors_bins(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
if length(pos_estimated) ~= length(score)
   error('pos_estimated  and  score not compatible');
end
if length(pos_estimated) ~= length(pos)
   error('pos_estimated  and  pos not compatible');
end
if length(pos_estimated) ~= length(endbulges)
   error('pos_estimated  and  endbulges size not compatible');
end
N = 100;
Per_bin = 20;
mxscore = max(score);
mnscore = min(score);
dth = (mxscore- mnscore)/N;
thresh = mnscore:dth:mxscore;
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)-Per_bin
   I = find(score >= thresh(i) & score <= thresh(i+Per_bin));
   if ~isempty(I)
      count = count + 1;
      midbin(count) = 0.5*(thresh(i) + thresh(i+Per_bin));
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
      correct_side_disth(count) = length(Jh)/length(I);


      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
```

```
      midbin(count) = 0.25*(thresh(i) + 2* thresh(i+1) + thresh(i+2));
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
    end
  end

  acc1 = accuracy + correct_side_dist1;
  acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
  clf
  hold on

  plot(midbin, acc2,'g')
  plot(midbin, acc1,'r')
  plot(midbin, accuracy,'b')
  plot(midbin, wrong_side,'k')
  plot(midbin,fraction,'c')

  legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side');
  plot(midbin, acc2,'*g')
  plot(midbin, acc1,'or')
  plot(midbin, accuracy,'bd')
  plot(midbin, wrong_side,'kv')
  xlabel('bin');
  %keyboard
  returnfunction analyse_errors_bins1(pos_estimated,score,pos, endbulges,N)
%analyse_errors_bins1(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
if length(pos_estimated) ~= length(score)
    error('pos_estimated  and  score not compatible');
end
if length(pos_estimated) ~= length(pos)
    error('pos_estimated  and  pos not compatible');
end
if length(pos_estimated) ~= length(endbulges)
    error('pos_estimated  and  endbulges size not compatible');
end
if nargin == 4
   N = 6;
end
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
```

```
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)-1
   I = find(score <= thresh(i) & score >= thresh(i+1));
   if ~isempty(I)
      count = count + 1;
      midbin(count) = mean(score(I));
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
      correct_side_disth(count) = length(Jh)/length(I);


      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
      midbin(count) = NaN;;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
   end
end

acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
clf
hold on

plot(midbin, acc2,'g')
plot(midbin, acc1,'r')
plot(midbin, accuracy,'b')
plot(midbin, wrong_side,'k')
plot(midbin,fraction,'c')

legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side');
plot(midbin, acc2,'*g')
```

```
 plot(midbin, acc1,'or')
 plot(midbin, accuracy,'bd')
 plot(midbin, wrong_side,'kv')
 xlabel('bin');
 %keyboard
 returnfunction analyse_errors_perc(pos_estimated,score,pos, endbulges)
%analyse_errors_perc(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
N = 100;
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
   I = find(score > thresh(i));
   if ~isempty(I)
      count = count + 1;
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
      correct_side_disth(count) = length(Jh)/length(I);


      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
   end
 end
```

```
 acc1 = accuracy + correct_side_dist1;
 acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
 clf
 hold on

 plot(perc, acc2,'g')
 plot(perc, acc1,'r')
 plot(perc, accuracy,'b')
 plot(perc, wrong_side,'k')
 plot(perc, thresh,'c')
 legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'threshold');
 xlabel('percentage');
 axis([0 100 0 1]);
 %keyboard
 returnfunction analyse_errors_thresh(pos_estimated,score,pos, endbulges)
%analyse_errors_thresh(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
if max(score) > 1
   mxscore = max(score);
else
   mxscore = 1;
end
if min(score) < 0
   mnscore = min(score);
else
   mnscore = 0;
end
Np = 500;
dth = (mxscore- mnscore)/Np;
thresh = mnscore:dth:mxscore;
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
   eb = find(endbulges{i});
   correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
    I = find(score > thresh(i));
    if ~isempty(I)
       count = count + 1;
       accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

       J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
```

```
    correct_side_dist1(count) = length(J1)/length(I);
    J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
    correct_side_dist2(count) = length(J2)/length(I);
    Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
    correct_side_disth(count) = length(Jh)/length(I);


    wrong_side(count) = sum(1-correct_side(I))/length(I);

    fraction(count) = length(I)/N;
  else
    count = count+1;
    accuracy(count) = NaN;
    correct_side_dist1(count) = NaN;
    correct_side_dist2(count) = NaN;
    correct_side_disth(count) = NaN;
    wrong_side(count) = NaN;
    fraction(count) = NaN;
  end
end

acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
clf
hold on

plot(thresh, acc2,'g')
plot(thresh, acc1,'r')
plot(thresh, accuracy,'b')
plot(thresh, wrong_side,'k')
plot(thresh, fraction,'c')
legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'fraction');
xlabel('threshold');
%keyboard
returnfunction y = edit_distance(s,t)
% y = edit_distance(s,t)
%  compute edit (levenstein) distance between s and t
C = 0.5; % parameter that fixes the relative
%Algorithm
%
%Construct a matrix containing 0..m rows and 0..n columns.
% Initialize the first row to 0..n.
% Initialize the first column to 0..m.
% 3. Examine each character of s (i from 1 to n).
% 4. Examine each character of t (j from 1 to m).
% 5. If s[i] equals t[j], the cost is 0.
%6. If s[i] doesn't equal t[j], the cost is 1.
% Set cell d[i,j] of the matrix equal to the minimum of:
%a. The cell immediately above plus 1: d[i-1,j] + 1.
%b. The cell immediately to the left plus 1: d[i,j-1] + 1.
```

```
%c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.
%7 After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m
n = length(s);
m = length(t);
if n == 0
 y = m;
 return;
end
if m == 0
 y = n;
 return;
end
d = zeros(n+1,m+1); %Construct a matrix containing 0..m rows and 0..n columns.
d(1,:) = [0:m];       % Initialize the first row to 0..n.
d(:,1) = [0:n]';      %Initialize the first column to 0..m.
for i = 1:n
 for j = 1:m
  cost = (s(i) ~= t(j));
  d(i+1,j+1) = min([d(i+1,j)+1, d(i,j+1)+1 , d(i,j)+cost]);
 end
end
y = d(n+1,m+1);
return

function [pos,score] = edit_predict(seqsd, seqs, endbulges)
% y = edit_predict(seqsd, seqs, endbulges)
% find the best matching dicer position by its edit distance to one of the existing dicers
%
% GD 20.2
global Min_dlength  Alpha Step
paramfile = 'edit_params';
%addpath('d:/matlab'); % whereabouts of edit_distance
disp('calculating...');
Step = 1;
fid = fopen(paramfile,'r');
while ~feof(fid)
   line = fgetl(fid);
   eval(line)
end
fclose(fid);
for i = 1:length(seqs)
   %disp(num2str(i));
   [posi, scorei] = edit_predict1(seqsd,seqs{i}, endbulges{i});
   pos(i) = posi;
   score(i) = scorei;
end
return
function [pos, score] = edit_predict1(seqsd,seqsi, endbulgesi);
%calculate the best matching position of dicer
global Min_dlength Alpha Step
```

```
seq_size = length(seqsi);
lb = find(endbulgesi);
eb_size = length(lb);
eb_begin = lb(1);
eb_end = lb(eb_size);
nd = length(seqsd); % number of known dicers
length_seqi = length(seqsi);
%initialize variables with the largest possible distance
min_d = ones(length_seqi,1)*Min_dlength;
mean_d = ones(length_seqi,1)*Min_dlength;
%upper side
  for i = eb_begin-Min_dlength:-Step:1
  p = seqsi(i:i+Min_dlength-1);
  for j = 1:length(seqsd)
    % d(j) = edit_distance(p,seqsd{j});
    d(j) = editD(p,seqsd{j});
%     d(j) = editD(p,seqsd{j});
  end
  min_d(i) = min(d);
  % take also the mean of highest percentile
  [ds,l] = sort(d);
  mean_d(i) = mean(ds(1:floor(Alpha*nd)));
end
for i = eb_end+1:Step:length(seqsi)-Min_dlength+1
  p = seqsi(i:i+Min_dlength-1);
  for j = 1:length(seqsd)
    d(j) = editD(p,seqsd{j});
  end
  min_d(i) = min(d);
  % take also the mean of highest ten percentile
  [ds,l] = sort(d);
  mean_d(i) = mean(ds(1:floor(Alpha*nd)));
end
mmn = min(min_d);
l = find(min_d == mmn);
if length(l) ==1
  pos = l;
  score = Min_dlength - mmn;
else
  % take the position with hte highest alpha score
  [mn,J] = min(mean_d(l));
  pos = l(J);
  score = Min_dlength - mmn;
end
return
function [pos,score] = edit_predictk(seqsd, seqs, endbulges, k, thresh)
% y = edit_predictk(seqsd, seqs, endbulges, k, thresh);
% find the best matching dicer position by its edit distance to one of the existing dicers
% criterion is mean among best k matches
%thresh is the
```

```
% GD 20.2
global Min_dlength Step
paramfile = 'edit_params';
addpath('d:/matlab'); % whereabouts of edit_distance
if nargin <= 4
  thresh = 1.1;
end
if length(seqs) ~= length(endbulges)
  error('size of seqs and endbulges not campatible');
end
if thresh < 1
  error('thresh must be < 1');
end
Step = 1;
fid = fopen(paramfile,'r');
while ~feof(fid)
  line = fgetl(fid);
  eval(line)
end
fclose(fid);
for i = 1:length(seqs)
  disp(num2str(i));
  [posi, scorei] = edit_predict1(seqsd,seqs{i}, endbulges{i},k,thresh);
  pos(i) = posi;
  score(i) = scorei;
end
return
function [pos, score] = edit_predict1(seqsd,seqsi, endbulgesi,k,thresh);
%calculate the best matching position of dicer
global Min_dlength Step
seq_size = length(seqsi);
lb = find(endbulgesi);
eb_size = length(lb);
eb_begin = lb(1);
eb_end = lb(eb_size);
nd = length(seqsd); % number of known dicers
length_seqi = length(seqsi);
%initialize variables with the largest possible distance
min_d = ones(length_seqi,1)*Min_dlength;
mean_d = ones(length_seqi,1)*Min_dlength;
%upper side
for i = eb_begin-Min_dlength:-Step:1
  p = seqsi(i:i+Min_dlength-1);
  for j = 1:length(seqsd)
    d(j) = edit_distance(p,seqsd{j});
  end
  % take also the mean of best k
  [ds,l] = sort(d);
  mean_d(i) = mean(ds(1:k));
end
```

```
%lower side
for i = eb_end+1:Step:length(seqsi)-Min_dlength+1
   p = seqsi(i:i+Min_dlength-1);
   for j = 1:length(seqsd)
      d(j) = edit_distance(p,seqsd{j});
   end
   [ds,I] = sort(d);
   mean_d(i) = mean(ds(1:k));
end
mmn = min(mean_d);
I = find(mean_d <= thresh* mmn);
if length(I) ==1
   pos = I;
   score = Min_dlength - mmn;
else
   % take the position closest to loop
   side = sign(I - eb_begin);
   loopdist = 0.5*(1-side).* (eb_begin - I - Min_dlength) + 0.5*(1+side).* (I- eb_end-1);
   [mndist,J] = min(loopdist);
   I = I(J);
   pos = I;
   score = Min_dlength - mean_d(I);
end
return
function [si,sj] = find_identical_pairs(seqs)
%[si,sj] = find_identical_pairs(seqs)
% find identical palindromes in list
L = length(seqs);
for i = 1:L
   lenp(i) = length(seqs{i});
end
[lenps, I] = sort(lenp);
seqs = seqs(I);
count = 0;
for i = 1:L
   for j = i+1:L
      if lenps(i) ~= lenps(j)
         break
      else
         if all(seqs{i} == seqs{j})
            count = count+1;
            si(count) = I(i);
            sj(count) = I(j);
         end
      end
   end
end

      function strseq = int2nuc(intseq, ncase)
%strseq = int2nuc(intseq, ncase)
```

```
%convert a sequence of '1 2 3 4' into  'A C T G'  or 'a c t g'
% ncase = uppercase | lowercase
if nargin == 1
   ncase = 'uppercase';
end
if strcmp(ncase,'uppercase')
   nucs = 'ACTG';
elseif strcmp(ncase,'lowercase')
   nucs = 'actg';
end
strseq = char(size(intseq));
for i = 1:length(intseq)
   strseq(i) = nucs(intseq(i));
end
return
method = 'poly3'
if method == 'poly2'
% poly2 combined
    %configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
    %points on side error line
    as = [-1.5000   -0.9988   -0.5012   -0.0035    0.5012    0.99    1.4927];
    bs = [0.2325    0.2295    0.1360    0.0804    0.0336    0.0102    0.0015];
%points on precise within2 line
    ap = [-1.4965   -0.9988   -0.5012    0.0035    0.5012    0.9988];
    bp = [0.6798    0.6974    0.8348    0.9196    0.9722    0.9985];
elseif method == 'poly3'
    % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
    % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
     %points on side error line
    as = [-1.4981   -1.2412   -0.9994   -0.5006    0.0019    0.5044    0.7727    0.9994    1.2714    1.5057];
    bs = [0.2303    0.2248    0.2028    0.1239    0.0872    0.0560    0.0211    0.0211         0         0];
    %points on precise within2 line
    ap = [ -1.4981   -1.3281   -1.0031   -0.4931    0.0019    0.4969    0.9994    1.5000];
    bp = [0.6940    0.7000    0.7312    0.8688    0.9165    0.9404    0.9752    1.0000
yside = 1-interp1(as,bs,xi,'linear','extrap')
yprec = interp1(ap,bp,xi,'linear','extrap')
function [yside, yprec2] = interpolate_prob_new(score, fitfile);
%[yside, yprec2] = interpolate_prob_new(score, fitfile);
% load the parameters for interpolation
fid = fopen(fitfile,'r');
while ~feof(fid)
   line = fgetl(fid);
   if ~isstr(line), break , end;
   eval(line)
end
fclose(fid);
%interpolate
yside = interp1(xs,ys,score,'linear');
yprec2 = interp1(xp2,yp2,score,'linear');
returnfunction [yside, yprec2] = interpolate_prob_old_ver5(xi, method);
```

```
%[yside, yprec2] = interpolate_prob_old_ver5(yi, method);
% parameters are configuration specific see below!
disp('these are the accumulated performance, not the actual performance per bin');
disp(' press enter to continue');
pause
if nargin == 1
   method = 'poly3';
end
if strcmp(method, 'poly2')
% poly2 combined
   %configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   %points on side error line
   as = [-1.5000   -0.9988   -0.5012   -0.0035    0.5012    0.99    1.4927];
   bs = [0.2325    0.2295    0.1360    0.0804    0.0336    0.0102    0.0015];
%points on precise within2 line
   ap = [-1.4965   -0.9988   -0.5012    0.0035    0.5012    0.9988];
   bp = [0.6798    0.6974    0.8348    0.9196    0.9722    0.9985];
elseif strcmp(method,'poly3')
   % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
    %points on side error line
   as = [-1.4981   -1.2412   -0.9994   -0.5006    0.0019    0.5044    0.7727    0.9994    1.2714    1.5057];
   bs = [0.2303    0.2248    0.2028    0.1239    0.0872    0.0560    0.0211    0.0211         0         0];
   %points on precise within2 line
   ap = [ -1.4981   -1.3281   -1.0031   -0.4931    0.0019    0.4969    0.9994    1.5000];
   bp = [0.6940    0.7000    0.7312    0.8688    0.9165    0.9404    0.9752    1.0000];
end
yside = 1-interp1(as,bs,xi,'linear');
yprec2 = interp1(ap,bp,xi,'linear');
function [yside, yprec2] = interpolate_probabilities(xi, method);
%[yside, yprec2] = interpolate_probabilities(yi, method);
% parameters are configuration specific see below!
if nargin == 1
   method = 'poly3';
end
if strcmp(method,'poly3')
   % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
    %points on side error line
 as = [1.5000  1.3235  1.2591  0.9478  0.2052  -0.1707  -0.3337  -0.5573  -0.7706  -1.0873];
 bs = [1.0000  1.0000  0.9355  0.8710  0.8485  0.8438  0.8182  0.5806  0.5000  0.5000];
 ap = as;
 bp = [1.0000  1.0000  0.9355  0.8710  0.8485  0.8438  0.7576  0.4839  0.2803  0.2681];
end
yside = 1-interp1(as,bs,xi,'linear','extrap');
yprec2 = interp1(ap,bp,xi,'linear','extrap');
function [yside, yprec2] = interpolate_probabilities_ver5(xi, method);
%[yside, yprec2] = interpolate_probabilities_ver5(yi, method);
% parameters are configuration specific see below!
disp('these are the accumulated performance, not the actual performance per bin');
```

```
disp(' press enter to continue');
pause
if nargin == 1
   method = 'poly3';
end
if strcmp(method, 'poly2')
% poly2 combined
   %configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   %points on side error line
   as = [-1.5000   -0.9988   -0.5012   -0.0035    0.5012    0.99    1.4927];
   bs = [0.2325    0.2295    0.1360    0.0804    0.0336    0.0102    0.0015];
%points on precise within2 line
   ap = [-1.4965   -0.9988   -0.5012    0.0035    0.5012    0.9988];
   bp = [0.6798    0.6974    0.8348    0.9196    0.9722    0.9985];
elseif strcmp(method,'poly3')
   % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
    %points on side error line
   as = [-1.4981   -1.2412   -0.9994   -0.5006    0.0019    0.5044    0.7727    0.9994    1.2714    1.5057];
   bs = [0.2303    0.2248    0.2028    0.1239    0.0872    0.0560    0.0211    0.0211         0         0];
   %points on precise within2 line
   ap = [ -1.4981   -1.3281   -1.0031   -0.4931    0.0019    0.4969    0.9994    1.5000];
   bp = [0.6940    0.7000    0.7312    0.8688    0.9165    0.9404    0.9752    1.0000];
end
yside = 1-interp1(as,bs,xi,'linear');
yprec2 = interp1(ap,bp,xi,'linear');
function [yside, yprec2] = interpolate_probabilities_ver5(xi, method);
%[yside, yprec2] = interpolate_probabilities_ver5(yi, method);
% parameters are configuration specific see below!
disp('these are the accumulated performance, not the actual performance per bin');
disp(' press enter to continue');
pause
if nargin == 1
   method = 'poly3';
end
if strcmp(method, 'poly2')
% poly2 combined
   %configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   %points on side error line
   as = [-1.5000   -0.9988   -0.5012   -0.0035    0.5012    0.99    1.4927];
   bs = [0.2325    0.2295    0.1360    0.0804    0.0336    0.0102    0.0015];
%points on precise within2 line
   ap = [-1.4965   -0.9988   -0.5012    0.0035    0.5012    0.9988];
   bp = [0.6798    0.6974    0.8348    0.9196    0.9722    0.9985];
elseif strcmp(method,'poly3')
   % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
    %points on side error line
   as = [-1.4981   -1.2412   -0.9994   -0.5006    0.0019    0.5044    0.7727    0.9994    1.2714    1.5057];
   bs = [0.2303    0.2248    0.2028    0.1239    0.0872    0.0560    0.0211    0.0211         0         0];
```

```
    %points on precise within2 line
    ap = [ -1.4981   -1.3281   -1.0031   -0.4931    0.0019    0.4969    0.9994    1.5000];
    bp = [0.6940    0.7000    0.7312    0.8688    0.9165    0.9404    0.9752    1.0000];
end
yside = 1-interp1(as,bs,xi,'linear');
yprec2 = interp1(ap,bp,xi,'linear');
function [yside, yprec2] = interpolate_probabilities_ver5(xi, method);
%[yside, yprec2] = interpolate_probabilities_ver5(yi, method);
% parameters are configuration specific see below!
disp('version 5 does not allow for extrapolation')
if nargin == 1
    method = 'poly3';
end
if strcmp(method,'poly3')
    % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
    % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
     %points on side error line
 as = [1.5000  1.3235  1.2591  0.9478  0.2052  -0.1707  -0.3337  -0.5573  -0.7706  -1.0873];
 bs = [1.0000  1.0000  0.9355  0.8710  0.8485  0.8438  0.8182  0.5806  0.5000  0.5000];
 ap = as;
 bp = [1.0000  1.0000  0.9355  0.8710  0.8485  0.8438  0.7576  0.4839  0.2803  0.2681];
end
%yside = 1-interp1(as,bs,xi,'linear','extrap');
%yprec2 = interp1(ap,bp,xi,'linear','extrap');
yside = 1-interp1(as,bs,xi,'linear');
yprec2 = interp1(ap,bp,xi,'linear');
function len = length_seq(seqs);
%len = length_seq(seqs);
%calculate sequence length
for i = 1:length(seqs)
   len(i) = length(seqs{i});
end
returnl = find(lenp-pos >= 22);
for i = 1:length(l)
frstl(i) = seqs{l(i)}(pos(l(i)));
lastl(i,:) = seqs{l(i)}([20+pos(l(i)), 21+pos(l(i))]);
end %load training data
randomize = 1;
curdir = pwd;
cd d:/rosetta/data_new
load matlab_147_unique.mat
if randomize
 disp('performing randomized permutation');
 l = randperm(length(seqs));
 bulges1 = bulges1(l);
 bulges2 = bulges2(l);
 endbulges = endbulges(l);
 lend = lend(l);
 lenp = lenp(l);
 pos = pos(l);
```

```
 seq_id = seq_id(l);
 seqs = seqs(l);
   seqsd = seqsd(l);
end
cd(curdir) %load training data
randomize = 0;
curdir = pwd;
cd d:/rosetta/data_new
load matlab_147_unique.mat
if randomize
 disp('performing randomized permutation');
 l = randperm(length(seqs));
 bulges1 = bulges1(l);
 bulges2 = bulges2(l);
 endbulges = endbulges(l);
 lend = lend(l);
 lenp = lenp(l);
 pos = pos(l);
 seq_id = seq_id(l);
 seqs = seqs(l);
   seqsd = seqsd(l);
end
cd(curdir) %load training data
curdir = pwd;
cd d:/rosetta/data_new
load matlab_173_unique.mat
cd(curdir) function pos = locate_dicer(dicer_seq,pal_seq);
%pos = locate_dicer(dicer_seq,palseq)
%get absolute position of dicer on palindrom, from the beginning of the pllindrom
if length(dicer_seq) ~= length(pal_seq)
   error('different number of sequences');
end
pos = zeros(1,length(dicer_seq));
for i = 1:length(dicer_seq)
   l = findstr(dicer_seq{i}, pal_seq{i});
   if length(l) == 1
      pos(i) = l;
   else
      pos(i) = NaN;
   end
end
function pos_dummy = make_pos_dummy(seqs,bulges1,bulges2,endbulges)
%pos_dummy = make_pos_dummy(seqs,bulges1,bulges2,endbulges)
% construct dummy pos vector for testing classifiers
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
mode = 'testing'
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
pos_dummy = zeros(1, length(seqs));
for i = 1:length(seqs)
```

```
  pos_dummy(i) = mkpsi0(seqs{i},bulges1{i},bulges2{i},endbulges{i});
  %pos_dummy(i) = mkpsi1(seqs{i},bulges1{i},bulges2{i},endbulges{i});
end
return
% version 0
function posi = mkpsi0(seqsi,bulges1i,bulges2i,endbulgesi)
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
% simple rule
% assume dicer of length 17 exactly.
% nearest , in euclidean distance to some prototype, regardles of distance
% from loop and side
% params are assumed -2 3 -2 7
prototype = [0.3381,-0.4804,0.1813,-0.1205,0.3318,0.0028,0.2095,-0.3635, ...
  -0.0711,-0.1954,-0.3103,-0.3066,0.1822,-0.1972,0.0417,0.3385,-0.4882, ...
  -0.3491,0.1979,-0.1216,0.3600,0.3537,0.0936,0.2271,-0.1907,0.3939, ...
  0.3385,0.0681,-0.1296,0.2027,0.0466,0.2948,0.4568,0.0226,0.0182,...
  0.0828,-0.0765,0.0155,-0.1660,-0.0671,-0.2741,0.0798,-0.3252,0.0678 ...
  0.2604,0.0298,0.1405,-0.2909,-0.1202,0.2833,0.1808,-0.4104,-0.0389];
seq_size = length(seqsi);
lb = find(endbulgesi);
eb_size = length(lb);
eb_begin = lb(1);
[xi, yi] = preprocess5(seqsi,bulges1i,bulges2i,endbulgesi);
[m,n] = size(xi);
sim = xi(:,3:n)*(prototype(1:n-2))';
[maxs,m] = max(sim);
side = xi(m,1);
loopdist = xi(m,2) * (0.5* (seq_size - eb_size));
posi = (1+side)/2*(eb_end + loopdist) + (1-side)/2*(eb_begin - loopdist);
return
% version 1
function posi = mkpsi1(seqsi,bulges1i,bulges2i,endbulgesi)
% simple rule
% assume dicer of length 17 exactly.
% nearest position to loop, such that dicer begins with t , not on bulge1
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
lb = find(endbulgesi);
eb_size = length(lb);
eb_begin = lb(1);
eb_end = lb(eb_size);
pos = find(seqsi == 3 & endbulgesi == 0 & bulges1i == 0);
dst = zeros(size(pos));
if ~isempty(pos)
  side = sign(pos-eb_begin);
  lup = find(side == -1);
  dst(lup) = eb_begin - (pos(lup) + Min_dlength -1);
  ldwn = find(side == 1);
  dst(ldwn) = pos(ldwn) - eb_end;
```

```
   dst(find(dst < 0))= 1000;
   [md,l] = min(dst);
   posi = pos(l);
else
 pos = find(seqsi == 4 & endbulgesi == 0 & bulges1i == 0);
 side = sign(pos-eb_begin);
   lup = find(side == -1);
   dst(lup) = eb_begin - (pos(lup) + Min_dlength -1);
   ldwn = find(side == 1);
   dst(ldwn) = pos(ldwn) - eb_end;
   on_endbulge = find(dst < 0);
   dst(on_endbulge) == 1000;
   [md,l] = min(dst);
   posi = pos(l);
end
return
function [x,y,seqno] = merge_sets(x1,x2,y1,y2,seqno1,seqno2)
%[x,y,seqno] = merge_sets(x1,x2,y1,y2,seqno1,seqno2)
% concatenate datasets
x = [x1; x2];
y = [y1; y2];
seqno = [seqno1; seqno2+max(seqno1)];
return%mfold_cv
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
pos5 = zeros(0);
score5 =zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];% test set
   bt = setdiff(bins_all , bs);% train set

   disp(' ');
   disp([ 'm = ' num2str(m)]);

   [pos5m,score5m] = edit_predict(seqsd(bt), seqs(bs), endbulges(bs));

  pos5 = [pos5,pos5m];
  score5 =[score5,score5m];

  m = m+1;
end
% perform m fold cross validation on article + zuker results by splitting set
validation = 1; % otherwise, only testing is performed
mfold = 5;
n_all = 278;
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
```

```
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
pos3 = zeros(0);
score3 = zeros(0);
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
pos5 = zeros(0);
score5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set
   filename3 = ['svm_tst_3m.dat'];
   filename5 = ['svm_tst_5m.dat'];
   [x3s, seqno3s] = preprocess_and_write_data3(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename3);
   [x5s, seqno5s] = preprocess_and_write_data5(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename5);
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

   bt = setdiff(bins_all , bs);
   filename3 = ['svm_trn_3m.dat'];
   filename5 = ['svm_trn_5m.dat'];
   [x3t, seqno3t] = preprocess_and_write_data3(seqs_all(bt),bulges1_all(bt),bulges2_all(bt),endbulges_all(bt),
filename3, pos_all(bt)+lend_all(bt)-1);
   [x5t, seqno5t] = preprocess_and_write_data5(seqs_all(bt),bulges1_all(bt),bulges2_all(bt),endbulges_all(bt),
filename5, pos_all(bt));
   disp('written preprocessed training examples');

   disp('now train and test svm. results should be in   g:\research\rosetta\svm_light_utils1\svm_outputs\out3m.out,
out5m.out');
   disp(' ***********************************************************************************************************');

   pause
   cd svm_outputs
   load out3m.out
   load out5m.out
   cd ..
   [pos3m, score3m] = svm_position(x3s,out3m,seqno3s, endbulges_all(bs), lenp_all(bs)+lend_all(bs));
   [pos5m, score5m] = svm_position(x5s,out5m,seqno5s, endbulges_all(bs), lenp_all(bs));
   %collect global variables
   x3 = [x3; x3s];
   out3 =[out3;out3m];
   if m == 1
      seqno3 = seqno3s;
   else
      mx3 = max(seqno3);
```

```
        seqno3 = [seqno3 ; mx3+seqno3s];
    end
    pos3 = [pos3 pos3m];
    score3 = [score3 score3m];

    x5 = [x5;x5s];;
    out5 =[out5;out5m];
    if m == 1
        seqno5 = seqno5s;
    else
        mx5 = max(seqno5);
        seqno5 = [seqno5 ; mx3+seqno5s];
    end
    pos5 = [pos5 pos5m];
    score5 = [score5 score5m];
    m = m+1;
end
%mfold_cv_transduction
% perform m fold cross validation on article + zuker results by splitting set
% use transduction mode of SVM
mfold = 5;
n_all = 278;
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
pos3 = zeros(0);
score3 = zeros(0);
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
pos5 = zeros(0);
score5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
   %training set
   bt = setdiff(bins_all , bs);
   filename3 = ['svm_trn_3t.dat'];
   filename5 = ['svm_trn_5t.dat'];
   [x3t, seqno3t] = preprocess_and_write_data3(seqs_all(bt),bulges1_all(bt),bulges2_all(bt),endbulges_all(bt),
filename3, pos_all(bt)+lend_all(bt)-1);
   [x5t, seqno5t] = preprocess_and_write_data5(seqs_all(bt),bulges1_all(bt),bulges2_all(bt),endbulges_all(bt),
filename5, pos_all(bt));
   disp('written preprocessed training examples');

% test set - append to previous file
    [x3s, seqno3s] = preprocess_and_write_data3_tr(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename3);
```

```
   [x5s, seqno5s] = preprocess_and_write_data5_tr(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename5);

   % test set - write to seperate file
   filename3 = ['svm_tst_3t.dat'];
   filename5 = ['svm_tst_5t.dat'];
   [x3s, seqno3s] = preprocess_and_write_data3(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename3);
   [x5s, seqno5s] = preprocess_and_write_data5(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename5);
  disp('written preprocessed testing examples');
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

disp('now train and test svm. ');
disp('transductive data are in  svm_trn_5t.dat  etc.')
disp('results should be in   g:\research\rosetta\svm_light_utils1\svm_outputs\out5t.out, etc.');
disp(' ***************************************************************************************************************');

   pause
   cd svm_outputs
   load out3t.out
   load out5t.out
   cd ..
   [pos3t, score3t] = svm_position(x3s,out3t,seqno3s, endbulges_all(bs), lenp_all(bs)+lend_all(bs));
   [pos5t, score5t] = svm_position(x5s,out5t,seqno5s, endbulges_all(bs), lenp_all(bs));
   %collect global variables
   x3 = [x3; x3s];
   out3 =[out3;out3t];
   if m == 1
      seqno3 = seqno3s;
   else
      mx3 = max(seqno3);
      seqno3 = [seqno3 ; mx3+seqno3s];
   end
   pos3 = [pos3 pos3t];
   score3 = [score3 score3t];

   %here am


   x5 = [x5;x5s];;
   out5 =[out5;out5t];
   if m == 1
      seqno5 = seqno5s;
   else
      mx5 = max(seqno5);
      seqno5 = [seqno5 ; mx3+seqno5s];
   end
   pos5 = [pos5 pos5t];
```

```
    score5 = [score5 score5t];
    m = m+1;
end
% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename3 = 'd:/svm_light/model3m';
tst_filename3 = 'd:/rosetta/svm_light_utils1/svm_tst_3m.dat';
trn_filename3 = 'd:/rosetta/svm_light_utils1/svm_trn_3m.dat';
out_filename3 = 'd:/rosetta/svm_light_utils1/out3m.out';
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
m = 1;
while m <= mfold
    bs = [bins(m)+1: bins(m+1)];
% test set

[x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs), ...
   bulges2(bs),endbulges(bs), tst_filename3);
    disp([ 'm = ' num2str(m)]);
    disp('written preprocessed test examples');

    bt = setdiff(bins_all , bs);

    [x3t, seqno3t] = preprocess_and_write_data3(seqs(bt),bulges1(bt),...
       bulges2(bt),endbulges(bt), trn_filename3, pos(bt)+lend(bt)-1);
    disp('written preprocessed training examples');

   dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename3 ' ' model_filename3]);
    dos(['d:/svm_light/svm_classify ' tst_filename3 ' ' model_filename3 ' ' out_filename3]);

    load out3m.out

    %collect global variables
    x3 = [x3;x3s];
    out3 =[out3;out3m];
    if m == 1
        seqno3 = seqno3s;
    else
        mx3 = max(seqno3);
        seqno3 = [seqno3 ; mx3+seqno3s];
    end
    m = m+1;
end
clear x3s x3t out3m seqno3s seqno3t bs bt
% just for printing  the info
```

```
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
disp(' ');
disp('*******************************************************');
disp('3 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);
% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename3 = 'd:/svm_light/model3m';
tst_filename3 = 'd:/rosetta/svm_light_utils1/svm_tst_3mb.dat';
trn_filename3 = 'd:/rosetta/svm_light_utils1/svm_trn_3mb.dat';
out_filename3 = 'd:/rosetta/svm_light_utils1/out3mb.out';
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set

[x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs), ...
  bulges2(bs),endbulges(bs), tst_filename3);
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

   bt = setdiff(bins_all , bs);

   [x3t, seqno3t] = preprocess_and_write_data3(seqs(bt),bulges1(bt),...
      bulges2(bt),endbulges(bt), trn_filename3, pos(bt)+lend(bt)-1);
   disp('written preprocessed training examples');

  dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename3 ' ' model_filename3]);
   dos(['d:/svm_light/svm_classify ' tst_filename3 ' ' model_filename3 ' ' out_filename3]);

   load out3mb.out
   out3m= out3mb;

   %collect global variables
   x3 = [x3;x3s];;
   out3 =[out3;out3m];
   if m == 1
      seqno3 = seqno3s;
   else
      mx3 = max(seqno3);
      seqno3 = [seqno3 ; mx3+seqno3s];
   end
```

```
   m = m+1;
end
clear x3s x3t out3m out3mb seqno3s seqno3t bs bt
% just for printing  the info
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
disp(' ');
disp('*******************************************************');
disp('3 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);
% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename5 = 'd:/svm_light/model5m';
tst_filename5 = 'd:/rosetta/svm_light_utils1/svm_tst_5m.dat';
trn_filename5 = 'd:/rosetta/svm_light_utils1/svm_trn_5m.dat';
out_filename5 = 'd:/rosetta/svm_light_utils1/out5m.out';
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set

[x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs), ...
  bulges2(bs),endbulges(bs), tst_filename5);
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

   bt = setdiff(bins_all , bs);

   [x5t, seqno5t] = preprocess_and_write_data5(seqs(bt),bulges1(bt),...
      bulges2(bt),endbulges(bt), trn_filename5, pos(bt));
   disp('written preprocessed training examples');

  dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename5 ' ' model_filename5]);
   dos(['d:/svm_light/svm_classify ' tst_filename5 ' ' model_filename5 ' ' out_filename5]);

   load out5m.out

   %collect global variables
   x5 = [x5;x5s];;
   out5 =[out5;out5m];
   if m == 1
      seqno5 = seqno5s;
   else
```

```
        mx5 = max(seqno5);
        seqno5 = [seqno5 ; mx5+seqno5s];
    end
    m = m+1;
end
clear x5s x5t out5m seqno5s seqno5t bs bt
% just for printing  the info
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
disp(' ');
disp('*******************************************************');
disp('5 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);
% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename5 = 'd:/svm_light/model5m';
tst_filename5 = 'd:/rosetta/svm_light_utils1/svm_tst_5m.dat';
trn_filename5 = 'd:/rosetta/svm_light_utils1/svm_trn_5m.dat';
out_filename5 = 'd:/rosetta/svm_light_utils1/out5m.out';
model_filename3 = 'd:/svm_light/model3m';
tst_filename3 = 'd:/rosetta/svm_light_utils1/svm_tst_3m.dat';
trn_filename3 = 'd:/rosetta/svm_light_utils1/svm_trn_3m.dat';
out_filename3 = 'd:/rosetta/svm_light_utils1/out3m.out';
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
pos3 = zeros(0);
score3 = zeros(0);
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
pos5 = zeros(0);
score5 = zeros(0);
m = 1;
while m <= mfold
    bs = [bins(m)+1: bins(m+1)];
% test set
    [x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), tst_filename3);
    [x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), tst_filename5);
    disp([ 'm = ' num2str(m)]);
    disp('written preprocessed test examples');

    bt = setdiff(bins_all , bs);
    [x3t, seqno3t] = preprocess_and_write_data3(seqs(bt),bulges1(bt),bulges2(bt),endbulges(bt), trn_filename3,
pos(bt)+lend(bt)-1);
    [x5t, seqno5t] = preprocess_and_write_data5(seqs(bt),bulges1(bt),bulges2(bt),endbulges(bt), trn_filename5,
pos(bt));
```

```
    disp('written preprocessed training examples');

    dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename5 ' ' model_filename5]);
    dos(['d:/svm_light/svm_classify ' tst_filename5 ' ' model_filename5 ' ' out_filename5]);
     dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename3 ' ' model_filename3]);
    dos(['d:/svm_light/svm_classify ' tst_filename3 ' ' model_filename3 ' ' out_filename3]);

    load(out_filename3);
    load(out_filename5)

    %[pos3m, score3m] = svm_position(x3s,out3m,seqno3s, endbulges(bs), lenp(bs));
    %[pos5m, score5m] = svm_position(x5s,out5m,seqno5s, endbulges(bs), lenp(bs));
    %collect global variables
    x3 = [x3; x3s];
    out3 =[out3;out3m];
    if m == 1
        seqno3 = seqno3s;
    else
        mx3 = max(seqno3);
        seqno3 = [seqno3 ; mx3+seqno3s];
    end
%    pos3 = [pos3 pos3m];
%    score3 = [score3 score3m];

    x5 = [x5;x5s];;
    out5 =[out5;out5m];
    if m == 1
        seqno5 = seqno5s;
    else
        mx5 = max(seqno5);
        seqno5 = [seqno5 ; mx3+seqno5s];
    end
 %   pos5 = [pos5 pos5m];
 %   score5 = [score5 score5m];
    m = m+1;
end
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
disp(' ');
disp('*******************************************************');
disp('5 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
disp(' ');
disp('*******************************************************');
disp('3 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
```

```
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename5 = 'd:/svm_light/model5mb';
tst_filename5 = 'd:/rosetta/svm_light_utils1/svm_tst_5mb.dat';
trn_filename5 = 'd:/rosetta/svm_light_utils1/svm_trn_5mb.dat';
out_filename5 = 'd:/rosetta/svm_light_utils1/out5mb.out';
model_filename3 = 'd:/svm_light/model3mb';
tst_filename3 = 'd:/rosetta/svm_light_utils1/svm_tst_3mb.dat';
trn_filename3 = 'd:/rosetta/svm_light_utils1/svm_trn_3mb.dat';
out_filename3 = 'd:/rosetta/svm_light_utils1/out3mb.out';
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
pos3 = zeros(0);
score3 = zeros(0);
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
pos5 = zeros(0);
score5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set
   [x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), tst_filename3);
   [x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), tst_filename5);
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

   bt = setdiff(bins_all , bs);
   [x3t, seqno3t] = preprocess_and_write_data3(seqs(bt),bulges1(bt),bulges2(bt),endbulges(bt), trn_filename3,
pos(bt)+lend(bt)-1);
   [x5t, seqno5t] = preprocess_and_write_data5(seqs(bt),bulges1(bt),bulges2(bt),endbulges(bt), trn_filename5,
pos(bt));
   disp('written preprocessed training examples');

   dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename5 ' ' model_filename5]);
   dos(['d:/svm_light/svm_classify ' tst_filename5 ' ' model_filename5 ' ' out_filename5]);
    dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename3 ' ' model_filename3]);
   dos(['d:/svm_light/svm_classify ' tst_filename3 ' ' model_filename3 ' ' out_filename3]);

   load(out_filename3);
   load(out_filename5)
   out5m = out5mb;
   out3m = out3mb;

   %[pos3m, score3m] = svm_position(x3s,out3m,seqno3s, endbulges(bs), lenp(bs));
   %[pos5m, score5m] = svm_position(x5s,out5m,seqno5s, endbulges(bs), lenp(bs));
   %collect global variables
   x3 = [x3; x3s];
```

```
   out3 =[out3;out3m];
   if m == 1
      seqno3 = seqno3s;
   else
      mx3 = max(seqno3);
      seqno3 = [seqno3 ; mx3+seqno3s];
   end
%    pos3 = [pos3 pos3m];
%    score3 = [score3 score3m];

   x5 = [x5;x5s];;
   out5 =[out5;out5m];
   if m == 1
      seqno5 = seqno5s;
   else
      mx5 = max(seqno5);
      seqno5 = [seqno5 ; mx3+seqno5s];
   end
 %   pos5 = [pos5 pos5m];
 %   score5 = [score5 score5m];
   m = m+1;
end
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
disp(' ');
disp('*********************************************************');
disp('5 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
disp(' ');
disp('*********************************************************');
disp('3 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);% perform m fold cross validation on article + zuker results by splitting set
% modified file names for input/output so that can be run in parralel
mfold = 5;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename5 = 'd:/svm_light/model5m_b';
tst_filename5 = 'd:/rosetta/svm_light_utils1/svm_tst_5m_b.dat';
trn_filename5 = 'd:/rosetta/svm_light_utils1/svm_trn_5m_b.dat';
out_filename5 = 'd:/rosetta/svm_light_utils1/out5m_b.out';
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set
```

```
  [x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs), ...
     bulges2(bs),endbulges(bs), tst_filename5);
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

   bt = setdiff(bins_all , bs);

   [x5t, seqno5t] = preprocess_and_write_data5(seqs(bt),bulges1(bt),...
     bulges2(bt),endbulges(bt), trn_filename5, pos(bt));
   disp('written preprocessed training examples');

  dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename5 ' ' model_filename5]);
   dos(['d:/svm_light/svm_classify ' tst_filename5 ' ' model_filename5 ' ' out_filename5]);

   load out5m_b.out

   %collect global variables
   x5 = [x5;x5s];;
   out5 =[out5;out5m_b];
   if m == 1
      seqno5 = seqno5s;
   else
      mx5 = max(seqno5);
      seqno5 = [seqno5 ; mx5+seqno5s];
   end
   m = m+1;
end
clear x5s x5t out5m_b seqno5s seqno5t bs bt
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
disp(' ');
disp('*******************************************************');
disp('5 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);
%mfold_cvk
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
k = 4;
pos5 = zeros(0);
score5 =zeros(0);
m = 1;
while m <= mfold
  bs = [bins(m)+1: bins(m+1)];% test set
  bt = setdiff(bins_all , bs);% train set

  disp([ 'm = ' num2str(m)]);
```

```
   [pos5m,score5m] = edit_predictk(seqsd(bt), seqs(bs), endbulges(bs),k);

  pos5 = [pos5,pos5m];
  score5 =[score5,score5m];

  m = m+1;
end
function [intseq, fault_seq] = nuc2int4_new(strseq);
%[intseq, fault_seq] = nuc2int4_new(strseq)
%convert a sequence of 'A C T G' into a array of 1 2 3 4
intseq = zeros(size(strseq));
fault_seq = 0;
for i = 1:length(strseq)
   switch upper(strseq(i))
      case 'A' , intseq(i) = 1;
      case 'C' , intseq(i) = 2;
      case 'T' , intseq(i) = 3;
      case 'G' , intseq(i) = 4;
      otherwise , intseq = []; fault_seq = 1; break;
   end
end
function run_edit_distance()
infile='c:\editdistance\draw_file.dat';
outfile='c:\editdistance\dicer_res.dat';
cd \\rosetta4\Development\gideon\edit_dist
seqsd = cell(0);
ii=0
fid=fopen('seqsd','r');
while ~feof(fid)
   ii=ii+1;
   seqsd{ii}=fgetl(fid);

end
fclose(fid);
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
fidin
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
   disp('reading structure...');
   [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);
   [pos,score] = edit_predict(seqsd, seqs, endbulges)

%write to file
%seq_id0 is added so as to sequential order of sequence numbers
 seq_id = seq_id + seq_id0;
 res = [seq_id; pos; score];
   fprintf(fidout, '%d %d %g ', res);
   seq_id0 = max(seq_id);
```

```
end
fclose(fidin);
fclose(fidout);
quit
return;
function y = prctile(x,p);
%PRCTILE gives the percentiles of the sample in X.
%   Y = PRCTILE(X,P) returns a value that is greater than P percent
%   of the values in X. For example, if P = 50  Y is the median of X.
%
%   P may be either a scalar or a vector. For scalar P, Y is a row
%   vector containing Pth percentile of each column of X. For vector P,
%   the ith row of Y is the P(i) percentile of each column of X.
%   Copyright (c) 1993-98 by The MathWorks, Inc.
%   $Revision: 2.6 $  $Date: 1997/11/29 01:46:27 $
[prows pcols] = size(p);
if prows ~= 1 & pcols ~= 1
   error('P must be a scalar or a vector.');
end
if any(p > 100) | any(p < 0)
   error('P must take values between 0 and 100');
end
xx = sort(x);
[m,n] = size(x);
if m==1 | n==1
   m = max(m,n);
 if m == 1,
   y = x*ones(length(p),1);
   return;
 end
   n = 1;
   q = 100*(0.5:m - 0.5)./m;
   xx = [min(x); xx(:); max(x)];
else
   q = 100*(0.5:m - 0.5)./m;
   xx = [min(x); xx; max(x)];
end
q = [0 q 100];
y = interp1(q,xx,p);
function seqtable = prepare_seqtable(seqno_list);
% seqtable = prepare_seqtable(seqno);
%seqtable conatins for each seqno its starting location in example list
% and its end location
seqtable = zeros(max(seqno_list),2);
i = 1;
seqno = seqno_list(i);
while i <= length(seqno_list)
  seqtable(seqno,1) = i;
  while seqno_list(i) == seqno
    seqtable(seqno,2) = i;
```

```
      i = i+1;
      if i > length(seqno_list)
        break
      end
    end
    if i > length(seqno_list)
        break
    end
    seqno = seqno_list(i);
end
return



   function [x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges, filename, pos)
%[x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges,filename, pos+lend-1);
%[x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges,filename);   %testing mode
% notice that here pos is pos
%x12 are the first two elements of x (side , relative_loopdist)
%high level function for preparing data and writing for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
if nargin == 5
    mode = 'testing';
else
    mode = 'training';
end
x12 = zeros(0);
seqno = zeros(0);
fid = fopen(filename,'w');
for i = 1:length(seqsp)
   if strcmp(mode,'training')
      [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
   elseif strcmp(mode,'testing')
      [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},NaN);
   end
   write_examples(xi, yi, fid);
   x12 = [x12; xi(:,1:2)];
   seqno = [seqno; i*ones(size(xi,1),1)];

   if mod(i,100) == 0; i, end
end
fclose(fid);
return
   function [x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges, filename, pos)
%[x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges,filename, pos+lend-1);
%[x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges,filename);   %testing mode
% notice that here pos is pos
%x12 are the first two elements of x (side , relative_loopdist)
```

```
%high level function for preparing data and writing for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
if nargin == 5
    mode = 'testing';
else
    mode = 'training';
end
x12 = zeros(0);
seqno = zeros(0);
fid = fopen(filename,'a');
for i = 1:length(seqsp)
  if strcmp(mode,'training')
    [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
  elseif strcmp(mode,'testing')
    [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},NaN);
  end
  write_examples(xi, yi, fid);
  x12 = [x12; xi(:,1:2)];
  seqno = [seqno; i*ones(size(xi,1),1)];

  if mod(i,100) == 0; i, end
end
fclose(fid);
return
  function [x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges, filename, pos)
%[x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges,filename, pos);
%[x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges,filename);   %testing mode
%x12 are the first two elements of x (side , relative_loopdist)
%high level function for preparing data and writing for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
if nargin == 5
    mode = 'testing';
else
    mode = 'training';
end
%Maxsize is a simple upper bound for the number of possible positions
Maxsize = 0;
for i = 1:length(seqsp);
  Maxsize = Maxsize+length(seqsp{i});
end
x12 = zeros(Maxsize,2);
seqno = zeros(Maxsize,1);
xfrom = 1; % index where to write into xi and seqno
fid = fopen(filename,'w');
for i = 1:length(seqsp)
```

```
  if strcmp(mode,'training')
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
  elseif strcmp(mode,'testing')
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},NaN);
  end

  write_examples(xi, yi, fid);
  xlength = size(xi,1);

  x12(xfrom: xfrom + xlength-1,:) = xi(:,1:2);
  seqno(xfrom: xfrom + xlength-1) = i*ones(xlength,1);
  xfrom = xfrom + xlength ;

  if mod(i,1000) == 0; disp(i); end
end
fclose(fid);
% remove the unneeded sapce in x12 and seqno
x12(xfrom:Maxsize,:) = [];
seqno(xfrom:Maxsize) = [];
return
function [x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges, filename, pos)
%[x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges,filename, pos);
%[x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges,filename);   %testing mode
%x12 are the first two elements of x (side , relative_loopdist)
%high level function for preparing data and writing for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
if nargin == 5
   mode = 'testing';
else
   mode = 'training';
end
x12 = zeros(0);
seqno = zeros(0);
fid = fopen(filename,'a');
for i = 1:length(seqsp)
  if strcmp(mode,'training')
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
  elseif strcmp(mode,'testing')
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},NaN);
  end

  write_examples(xi, yi, fid);
  x12 = [x12; xi(:,1:2)];
  seqno = [seqno; i*ones(size(xi,1),1)];

  if mod(i,100) == 0; i, end
end
fclose(fid);
```

```
return
function [x,y,seqno] = preprocess_data3(seqsp,bulges1,bulges2,endbulges,pos)
%[x,y,seqno] = preprocess_data3(seqsp,bulges1,bulges2,endbulges,pos);   % for training
%[x,y,seqno] = preprocess_data3(seqsp,bulges1,bulges2,endbulges);    % for testing
%
% 3' side of MiR
%
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Mindlength] = read_params('params3.dat');
if nargin == 5
  mode = 'training';
else
  mode = 'testing';
end
x = zeros(0);
y = zeros(0);
seqno = zeros(0);
if strcmp(mode,'training')
  for i = 1:length(seqsp)
    [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
    x = [x; xi];
    y = [y; yi];
    seqno = [seqno; i*ones(size(yi))];
    if mod(i,10) == 0; i, end
  end
else
  for i = 1:length(seqsp)
    [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i});
    x = [x; xi];
    y = [y; yi];  % this is just a list of zeros
    seqno = [seqno; i*ones(size(yi))];
    if mod(i,10) == 0; i, end
  end
end
returnfunction [x,y,seqno] = preprocess_data5(seqsp,bulges1,bulges2,endbulges,pos)
%[x,y,seqno] = preprocess_data5(seqsp,bulges1,bulges2,endbulges,pos); % for training
%[x,y,seqno] = preprocess_data5(seqsp,bulges1,bulges2,endbulges); % for testing
%high level function for preparing data for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
Nnucfrom = -2; % nucleotides region of interest
Nnucto = 7;
Nbfrom = -2; %bulges region of interest
Nbto = 6;
Min_dlength = 17; % min dicer length
if nargin == 5
  mode = 'training';
else
  mode = 'testing';
```

```
end
x = zeros(0);
y = zeros(0);
seqno = zeros(0);
if strcmp(mode,'training')
  for i = 1:length(seqsp)
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
    x = [x; xi];
    y = [y; yi];
    seqno = [seqno; i*ones(size(yi))];
    if mod(i,10) == 0; i, end
  end
else
  for i = 1:length(seqsp)
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i});
    x = [x; xi];
    y = [y; yi];  % this is just a list of zeros
    seqno = [seqno; i*ones(size(yi))];
    if mod(i,10) == 0; i, end
  end
end
returnfunction x = preprocess_window(posj, seqwin,bulges1win,bulges2win, seq_size, eb_size, eb_begin, eb_end)
%preprocess_window : lower level function
% produces a feature vector from 3windows of the sequence
global Nnucfrom Nnucto Nbfrom Nbto mode
lenx = 2 + length(seqwin) *4 + 2*length(bulges1win);
x = zeros(0);
side = sign(posj-eb_begin);
x(1) = side; % -1 for upper, 1 for lower
loopdist = (1 + side)/2 * (posj - eb_end) + ...   % lower part
  (1 -side)/2* (eb_begin - posj);            % upper part
% normalize x2 by palyndrom available length
x(2) = loopdist/(0.5* (seq_size - eb_size));
n_assigned = 2;
binseq = zeros(4, Nnucto+1-Nnucfrom);
binseq([0:size(binseq,2)-1]*4 + seqwin) = 1;
binseq = binseq(:)';
x(n_assigned+1: n_assigned +length(binseq))  = binseq;
n_assigned = n_assigned + length(binseq);
x(n_assigned+1: n_assigned +2*length(bulges1win)) = [bulges1win bulges2win];
returnfunction [xi, yi] = preprocess3(seqspi,bulges1i,bulges2i,endbulgesi,posi)
% low level function aimed at processing a sigle sequence
%in testing mode, yi are simply 0;
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
seq_size = length(seqspi); % size of palindrome = # nucleotides
l = find(endbulgesi);
eb_size = length(l); % size of endbulge = loop
eb_begin = l(1);
eb_end = l(eb_size);
```

```
xi = zeros(0);
yi = zeros(0);
% range include for upper and lower 5' positions
from = min(Nbfrom,Nnucfrom);
to = max(Nbto, Nnucto);
for side = -1:2:1
  if side == -1
    posrange = Min_dlength : eb_begin-1-to;
  else
    posrange = eb_end+Min_dlength : seq_size-to;
  end

  for j = 1:length(posrange)
    posj = posrange(j);

    nuc_win = posj+Nnucfrom:posj+Nnucto;  %window of nucleotides (sequence)
    b_win = posj+Nbfrom:posj+Nbto;   %window of bulges (1 sided & 2 sided)


 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if length(seqspi) < max(nuc_win)
      disp('bug1')
    end
    if length(bulges1i) < max(b_win)| length(bulges2i) < max(b_win)
      disp('bug2')
    end
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    xij = preprocess_window(posj, seqspi(nuc_win), ...
        bulges1i(b_win),bulges2i(b_win), seq_size, eb_size, eb_begin, eb_end);
    xi = [xi; xij];
    if strcmp(mode,'training')
      yij = (posj == posi)*2-1; %+1 or -1
      yi = [yi ; yij];
    else
      yi = [yi ; 0];
    end
  end  % for j = 1:length(posrange)
end % if side ==
returnfunction [xi, yi] = preprocess5(seqspi,bulges1i,bulges2i,endbulgesi,posi)
% low level function aimed at processing a sigle sequence
%in testing mode, yi are simply 0;
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
%disp('preprocess5 modified. target id triangle like near 5 prime end');
seq_size = length(seqspi); % size of palindrome = # nucleotides
I = find(endbulgesi);
eb_size = length(I); % size of endbulge = loop
eb_begin = I(1);
eb_end = I(eb_size);
```

```
xi = zeros(0);
yi = zeros(0);
% range include for upper and lower 5' positions
from = min(Nbfrom,Nnucfrom);
to = max(Nbto, Nnucto);
for side = -1:2:1
  if side == -1
    posrange = 1+abs(from) :eb_begin-Min_dlength;
  else
    posrange = eb_end+1+abs(from) : seq_size+1-Min_dlength;
  end

  for j = 1:length(posrange)
    posj = posrange(j);

    nuc_win = posj+Nnucfrom:posj+Nnucto;  %window of nucleotides (sequence)
    b_win = posj+Nbfrom:posj+Nbto;   %window of bulges (1 sided & 2 sided)
    xij = preprocess_window(posj, seqspi(nuc_win), ...
        bulges1i(b_win),bulges2i(b_win), seq_size, eb_size, eb_begin, eb_end);
    xi = [xi; xij];

    if strcmp(mode,'training')
        yij = (posj == posi)*2-1; %+1 or -1
      % new version suitable for regression
    %yij = max(1-0.5*abs(posj-posi), -1);  % giving 1 at max and -1 at distance 3 or more
      yi = [yi ; yij];
    else
      yi = [yi ; 0];
    end
  end  % for j = 1:length(posrange)
end % if side ==
return[pos5,score5] = svm_position(x5,out5, seqno5, endbulges, lenp);
svkernel = input('enter kernel name: ','s');
targetdir = input('enter target directory name (e.g.) params-1-10-1-10: ' ,'s');
targetfile = ['d:\rosetta\svm_light_utils1\figures_174\' targetdir '\' svkernel];
figure(1); analyse_errors_thresh(pos5,score5,pos,endbulges); title(svkernel);
eval(['print -djpeg90 ' targetfile 'thresh']);
figure(2); analyse_errors_perc(pos5,score5,pos,endbulges); title(svkernel);
eval(['print -djpeg90 ' targetfile 'perc']);
in='c:\rosetta\data_baseline_15_5\draw_file2K.dat';
o = 'edist_res_file_hmdc257_2000pals.txt';
run_edit_distance_ranit(in,o);
in='c:\rosetta\data_baseline_15_5\200VirusesDraw.txt';
o = 'edist_res_file_hmdc257_virus.txt';
run_edit_distance_ranit(in,o);
in='c:\rosetta\data_baseline_15_5\badPalsGrade.txt';
o = 'edist_res_file_hmdc257_lowpal.txt';
run_edit_distance_ranit(in,o);
in='c:\rosetta\data_baseline_15_5\goodPalsGrade33.txt';
o = 'edist_res_file_hmdc257_highpal.txt';
```

```
run_edit_distance_ranit(in,o);function [Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params(paramsfile);
%[Nnucfrom, Nnucto, Nbfrom, Nbto, Mindlength] = read_params(paramsfile);
Nfields = 5;
fieldnames = cell(Nfields);
fieldnames(1:Nfields) = {'Nnucfrom'; 'Nnucto'; 'Nbfrom'; 'Nbto' ; 'Min_dlength'};
fid = fopen(paramsfile,'r');
while ~feof(fid)
   line = fgetl(fid);
   [field, rest] = strtok(line);

   if ~isempty(rest)
      value = num2str(strtok(rest));
   else
      error(['value of ' field  '  not specified']);
   end

   % assign the value to the proper variable
   found = 0;
   for i = 1:Nfields
      if strcmp(field, fieldnames{i})
         eval([field '=' num2str(value) ';']);
         found = 1;
         break
      end
   end
   if found == 0
      error(['illegal field  ' field  ]);
   end
end
fclose(fid);
return

function [seqs,len] = read_seq(filename);
%[len,seqs] = read_seq(filename);
%reads dicer or pal sequences into cell array, in numeric format
fid = fopen(filename,'r');
if fid == -1
  error([' file ' filename '  could not be opened']);
end
id = 0;
seq_no = 0;
while ~feof(fid)
  line = fgetl(fid);
  line = deblank(line);
  [intseq, fault_seq] = nuc2int4_new(line);
  id = id + 1;
  if fault_seq == 0
    seq_no = seq_no + 1;
    seqs{seq_no} = intseq;
    len(seq_no) = length(intseq);
```

```
  else
    disp(['faulty seq on id  ' num2str(id)])
  end

  if(mod(seq_no,1000) == 0 & seq_no ~= 0)
     disp(['seq_no ' num2str(seq_no)]);
  end
end
fclose(fid);
return

  function [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure(filename);
%[seqs,bulges1,bulges2,endbulges,seq_id] = read_structure(filename)
% read zuker structure
% seq is a cell array containing sequences
% bulge1 is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge2 is similarly for 2 sided bulge
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
if nargin == 0
  filename = 'C:\rosetta_versions\ver9\data\zuker_draw_z.txt';
end
fid = fopen(filename,'r');
seq_no = 0;
seqs = cell(0);
bulges1= cell(0);
bulges2= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
id = 0;
while ~feof(fid)
  structure = char(4,250);
  for i = 1:4
    line = fgetl(fid);
    structure(i,1:length(line)) = line;
  end
  id = id +1;
  [seqi, bulge1i, bulge2i, endbulgei] = get_features(structure);
  [intseq, fault_seq] = nuc2int4_new(seqi);
  if fault_seq == 0
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      bulges1{seq_no} = bulge1i;
      bulges2{seq_no} = bulge2i;
      endbulges{seq_no} = endbulgei;
      seq_id(seq_no) = id;
   else
      disp(['faulty seq on id  ' num2str(id)])
   end
  if(mod(seq_no,1000) == 0)
```

```
      seq_no
    end
  end
  fclose(fid);
  return
  function [seq, bulge1, bulge2, endbulge] = get_features(structure)
  % get sequence as well as bulge structure
  %upper half (5' side)
  bulge_row = 1; % the row of bulge letters
  bulge_row_opposite = 4;
  uphalf = structure(1:2,:);
  [j,k] = find(isletter(uphalf));
  max_col = max(k);
  count = 0;
  for col =1: max_col
    fl = find(isletter(uphalf(:,col)));
    if ~isempty(fl)
      count = count + 1;
      seq(count) = uphalf(fl,col);
      bulge = (fl == bulge_row);
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
        bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
        bulge1(count) = 1;
      end
    end
  end
  % endbulge is coded on the upper half
  % go backwards form 3' side to 5' side
  endbulge = zeros(size(bulge1));
  pos = length(bulge1);
  while bulge1(pos) == 1
    endbulge(pos) = 1;
    bulge1(pos) = 0;
    pos = pos - 1;
  end
  %lower half
  bulge_row = 2; % 4 th line on structure is 2 line on lower half
  bulge_row_opposite = 1;
  lwhalf = structure(3:4,:);
  [j,k] = find(isletter(lwhalf));
  max_col = max(k);
  for col =max_col:-1:1
    fl = find(isletter(lwhalf(:,col)));
    if ~isempty(fl)
      count = count + 1;
      seq(count) = lwhalf(fl,col);
      bulge = (fl == bulge_row);
```

```
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
return

  function [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fid,seqtot);
%[seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fid,seqtot)
% file id version: read 'seqtot' zuker draw palindromes from file handle 'fid'
%
% read zuker structure
% seq is a cell array containing sequences
% bulge1 is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge2 is similarly for 2 sided bulge
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
seq_no = 0;
seqs = cell(0);
bulges1= cell(0);
bulges2= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
id = 0;
while ~feof(fid) & seq_no < seqtot
  structure = char(4,250);
  for i = 1:4
    line = fgetl(fid);
    structure(i,1:length(line)) = line;
  end
  id = id +1;
  [seqi, bulge1i, bulge2i, endbulgei] = get_features(structure);
  [intseq, fault_seq] = nuc2int4_new(seqi);
  fault_structure = check_structure(seqi, bulge1i, bulge2i, endbulgei);

  if fault_seq == 0 & fault_structure == 0
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      bulges1{seq_no} = bulge1i;
      bulges2{seq_no} = bulge2i;
      endbulges{seq_no} = endbulgei;
      seq_id(seq_no) = id;
```

```
    else
        disp(['faulty seq on id  ' num2str(id)])
    end
   if(mod(seq_no,1000) == 0)
      seq_no
   end
end
return
function [seq, bulge1, bulge2, endbulge] = get_features(structure)
% get sequence as well as bulge structure
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
count = 0;
for col =1: max_col
   fl = find(isletter(uphalf(:,col)));
   if ~isempty(fl)
      count = count + 1;
      seq(count) = uphalf(fl,col);
      bulge = (fl == bulge_row);
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
         bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
         bulge1(count) = 1;
      end
   end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
pos = length(bulge1);
if(pos < 1)
   return
end
while bulge1(pos) == 1
   endbulge(pos) = 1;
   bulge1(pos) = 0;
   pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
lwhalf = structure(3:4,:);
[j,k] = find(isletter(lwhalf));
max_col = max(k);
```

```
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
return
function fault_structure = check_structure(seqi, bulge1i, bulge2i, endbulgei)
%test whether structure can be worked out by classifier, e.g.
% length of sequence is too short not enough space for Mir of length Mindlength
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
seq_size = length(seqi);
lb = find(endbulgei);
if(isempty(lb))
  fault_structure=1
  return
end
eb_size = length(lb);
eb_begin = lb(1);
eb_end = lb(eb_size);
% how many nucleotides/bulges are taken before 5' position
from = min(Nbfrom,Nnucfrom);
if (1+abs(from) > eb_begin-Min_dlength)  & ...
    (eb_end+1+abs(from) > seq_size+1-Min_dlength)
  fault_structure = 1;
else
  fault_structure = 0;
end
return

  function [seqs,bulges1,bulges2,endbulges,seq_id, conn] = read_structure_new(filename);
%[seqs,bulges1,bulges2,endbulges,seq_id,conn] = read_structure_new(filename)
% read zuker structure
% updated 22.1
% extractes also connection structure:
% conn(i) = index of nucleotide connected to nucleotide i (0 if unconnected);
% seq is a cell array containing sequences
% bulge1 is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge2 is similarly for 2 sided bulge
```

```
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
if nargin == 0
   filename = 'C:\rosetta_versions\ver9\data\zuker_draw_z.txt';
end
fid = fopen(filename,'r');
seq_no = 0;
seqs = cell(0);
bulges1= cell(0);
bulges2= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
conn = cell(0);
id = 0;
while ~feof(fid)
   structure = char(4,250);
   for i = 1:4
      line = fgetl(fid);
      structure(i,1:length(line)) = line;
   end
   id = id +1;
   [seqi, bulge1i, bulge2i, endbulgei, conni] = get_features(structure);
   [intseq, fault_seq] = nuc2int4_new(seqi);
   if fault_seq == 0
        seq_no = seq_no + 1;
        seqs{seq_no} = intseq;
        bulges1{seq_no} = bulge1i;
        bulges2{seq_no} = bulge2i;
        endbulges{seq_no} = endbulgei;
        seq_id(seq_no) = id;
        conn{seq_no} = conni;
    else
        disp(['faulty seq on id  ' num2str(id)])
    end
   if(mod(seq_no,1000) == 0)
      seq_no
   end
end
return
function [seq, bulge1, bulge2, endbulge, conn] = get_features(structure)
% get sequence as well as bulge structure
% sequence index of nucleotide in structure
structure_seq_ind = zeros(size(structure));
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
[j,k] = find(isletter(structure(1:2,:)));
max_col = max(k);
count = 0;
for col =1: max_col
```

```
  fl = find(isletter(structure(1:2,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = structure(1:2,col);
    bulge = (fl == bulge_row);
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    structure_seq_ind(col, fl) = count;
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 4; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(structure(3:4,:)));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(structure(3:4,col)));
  if ~isempty(fl)
    fl = fl+2; % add 2 since fl = 1/2 on structure(3:4,:
    count = count + 1;
    seq(count) = structure(fl,col);
    bulge = (fl == bulge_row);
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
    structure_seq_ind(col, fl) = count;
  end
end
% produce connection structure
conn = zeros(size(seq));
[j,k] = find(structure_seq_ind(2:3,:) ~= 0);
```

```
j_opp = 5-j; %opposite to j.    3 <-> 2
% produce connection matrix in simple representation
for i = 1:length(j)
  conn(structure_seq_ind(j(i),k(i))) = structure_seq_ind(j_opp(i),k(i));
end
return
function [seqsd, seqs,bulges1,bulges2,endbulges,seq_id] = remove_duplicates(seqsd,
seqs,bulges1,bulges2,endbulges,seq_id);
%[seqsd, seqs,bulges1,bulges2,endbulges,seq_id] = remove_duplicates(seqsd,
seqs,bulges1,bulges2,endbulges,seq_id);
% locate only unique palindrome-dicer pairs
% dicers must be sorted lexicographically
if length(seqsd) ~= length(seqs)
  error('seqsd  and seqs not compatible');
end
Idl = zeros(length(seqsd),1); %entries to be deleted
for i = 2:length(seqsd)
  if length(seqsd{i}) == length(seqsd{i-1}) & length(seqs{i}) == length(seqs{i-1})
    if all(seqsd{i} == seqsd{i-1}) & all(seqs{i} == seqs{i-1})
      Idl(i) = 1;
    end
  end
end
%delete duplicates
I = find(Idl);
seqsd(I) = [];
seqs(I) = [];
bulges1(I) = [];
bulges2(I) = [];
endbulges(I) = [];
seq_id(I) = [];
return

% perform a partitioned testing on large data
mfold = 3;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
m = 1;
fname = 'res_poly3_33156.out';
fid = fopen(fname, 'a');
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set
   filename3 = ['svm3_33156m.dat'];
   filename5 = ['svm5_33156m.dat'];

   [x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), filename3);
   [x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), filename5);
   disp([ 'm = ' num2str(m)]);
```

```
disp('written preprocessed test examples');

disp('now run svm_classify. inputs are svm3_33156m.dat and svm5_33156m.dat');
disp('results should be in   g:\research\rosetta\svm_light_utils1\svm_outputs\out3m.out, out5m.out');
disp(' ***************************************************************************************************************************');

pause
cd svm_outputs

% test that both out files exist
files_ok = 0;
while files_ok == 0;
   files_ok = 1;
   fid5 = fopen('out5m.out','r');
   fid3 = fopen('out3m.out','r');

   if fid5 == -1
      files_ok = 0;
      disp('run svm_classify on 3 data.  out3m.out not found. enter when ready');
      pause
   else
      fclose(fid5);
   end

   if fid3 == -1
      files_ok = 0;
      disp('run svm_classify on 3 data.  out3m.out not found. enter when ready');
      pause
   else
      fclose(fid3);
   end
end

load out3m.out
load out5m.out
%delete files to insure that on next iteration, files are new
delete out3m.out
delete out5m.out

cd ..

%[pos3m, score3m] = svm_position(x3s,out3m,seqno3s, endbulges(bs), lenp(bs));
%[pos5m, score5m] = svm_position(x5s,out5m,seqno5s, endbulges(bs), lenp(bs));
[pos53m, score53m] = svm_position53(x5s,out5m,seqno5s, x3s,out3m,seqno3s, endbulges(bs), lenp(bs));


[yside, yprec2] = interpolate_probabilities(score53m, 'poly3');

res =  [seq_id(bs); pos53m(:,1)'; pos53m(:,2)'; score53m'; yside'; yprec2'];
fprintf(fid, '%d %d %d %g %g %g\n', res);
```

```
    m = m+1;
end
fclose(fid);
function run_edit_distance()
%run_edit_distance(dicerfile, palfile, outfile)
fitfile = '\\rosetta4\Development\gideon\edit_dist\fit_21_025_1.txt'; %suitable for parameter alpha = 0.25
dicerfile='\\rosetta4\Development\gideon\edit_dist\seqsd'
palfile='c:\editdistance\draw_file.dat';
outfile='c:\editdistance\dicer_res.dat';
cd \\rosetta4\Development\gideon\edit_dist
[seqsd,len] = read_seq(dicerfile);
%transform to string
length(seqsd)
for i = 1: length(seqsd)
   seqsd{i} = int2nuc(seqsd{i},'uppercase');
end
fidin = fopen(palfile,'r');
fidout = fopen(outfile,'a');
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
   disp('reading structure...');
   [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);

   %transform back to string
   for i = 1: length(seqs)
      seqs{i} = int2nuc(seqs{i},'uppercase');
   end

   [pos,score] = edit_predict(seqsd, seqs, endbulges)

   %write to file
   %seq_id0 is added so as to sequential order of sequence numbers
   seq_id = seq_id + seq_id0;

   % interpolate
   [yside, yprec2] = interpolate_prob_new(score, fitfile);
   res = [seq_id; pos; score; yprec2;yside];
   fprintf(fidout, '%d %d %g %g %g ', res);
   seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
quit
function run_edit_distance()
infile='c:\editdistance\draw_file.dat';
outfile='c:\editdistance\dicer_res.dat';
cd \\rosetta4\Development\gideon\edit_dist
seqsd = cell(0);
```

```
ii=0
fid=fopen('seqsd','r');
while ~feof(fid)
  ii=ii+1;
  seqsd{ii}=fgetl(fid);

end
fclose(fid);
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
fidin
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
  disp('reading structure...');
  [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);
  [pos,score] = edit_predict(seqsd, seqs, endbulges)

%write to file
%seq_id0 is added so as to sequential order of sequence numbers
 seq_id = seq_id + seq_id0;
 res = [seq_id; pos; score];
  fprintf(fidout, '%d %d %g ', res);
  seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
quit
return;
function run_edit_distance_ranit(palfile,outfile)
fitfile = 'fit_21_025_1.txt'; %suitable for parameter alpha = 0.25
dicerfile='seqsd_hmdc257';
[seqsd,len] = read_seq(dicerfile);
%transform to string
length(seqsd)
for i = 1: length(seqsd)
  seqsd{i} = int2nuc(seqsd{i},'uppercase');
end
fidin = fopen(palfile,'r');
fidout = fopen(outfile,'w');
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
  disp('reading structure...');
  [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);

  %transform back to string
  for i = 1: length(seqs)
    seqs{i} = int2nuc(seqs{i},'uppercase');
  end
```

```
  [pos,score] = edit_predict(seqsd, seqs, endbulges)

  %write to file
  %seq_id0 is added so as to sequential order of sequence numbers
  seq_id = seq_id + seq_id0;

  % interpolate
  [yside, yprec2] = interpolate_prob_new(score, fitfile);
  res = [seq_id; pos; score; yprec2;yside];
  fprintf(fidout, '%d %d %g %g %g\n', res);
  seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
function [pos, score] = svm_position(x,svm_score,seqno, endbulges, lenp);
%[pos, score] = svm_position(x,svm_score, seqno, endbulges, lenp);
% postprocess svm outputs (for error analysis)
if size(x,1) ~= size(svm_score,1)
   error('x  and svm_score not compatible');
end
if size(x,1) ~= size(seqno,1)
   error('x  and seqno not compatible');
end
if max(seqno) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds = diff(seqno);
bnd = find(ds);
boundaries = [0 bnd' length(seqno)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
for s = 1:max(seqno)
   l = boundaries(s)+1 : boundaries(s+1);
   [maxs,m] = max(svm_score(l));
   score(s) = maxs;

   seq_size = lenp(s);
   lb = find(endbulges{s});
   eb_size = length(lb);
   eb_begin = lb(1);
   eb_end = lb(eb_size);

   side = x(l(m),1);
   loopdist = x(l(m),2) * (0.5* (seq_size - eb_size));
   pos(s) = (1+side)/2*(eb_end + loopdist) + (1-side)/2*(eb_begin - loopdist);
end
```

```
pos = round(pos);
returnfunction [pos, score] = svm_position_r(x,svm_score,seqno, endbulges, lenp);
%[pos, score] = svm_position_r(x,svm_score, seqno, endbulges, lenp);
% postprocess svm outputs (for error analysis)
% regression version
if size(x,1) ~= size(svm_score,1)
   error('x  and svm_score not compatible');
end
if size(x,1) ~= size(seqno,1)
   error('x  and seqno not compatible');
end
if max(seqno) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds = diff(seqno);
bnd = find(ds);
boundaries = [0 bnd' length(seqno)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
w = [-1.0  -0.5  0.0  0.5  1.0  0.5  0.0  -0.5  -1.0]; % window for convolution
nws = 0.5*(length(w)-1);
for s = 1:max(seqno)
  l = boundaries(s)+1 : boundaries(s+1);
  svm_scorel = svm_score(l);
  cnv = conv(w,svm_scorel);
  lcnv = length(cnv);
  % delete nws values on either side of cnv so that size equals that of scorel
  cnv([1:nws , lcnv-nws+1:lcnv]) = [];
   [maxs,m] = max(cnv);
   score(s) = maxs;

   seq_size = lenp(s);
   lb = find(endbulges{s});
   eb_size = length(lb);
   eb_begin = lb(1);
   eb_end = lb(eb_size);

   side = x(l(m),1);
   loopdist = x(l(m),2) * (0.5* (seq_size - eb_size));
   pos(s) = (1+side)/2*(eb_end + loopdist) + (1-side)/2*(eb_begin - loopdist);
end
pos = round(pos);
returnfunction [pos, score] = svm_position_soft(x,svm_score,seqno, endbulges, lenp);
%[pos, score] = svm_position_soft(x,svm_score, seqno, endbulges, lenp);
% postprocess svm outputs (for error analysis)
%
% takes the position closest to loop from positions which are at least
```

```
%  best score - (1-Thresh)*abs(best score)
%
Thresh = 0.8;
if size(x,1) ~= size(svm_score,1)
   error('x  and svm_score not compatible');
end
if size(x,1) ~= size(seqno,1)
   error('x  and seqno not compatible');
end
if max(seqno) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds = diff(seqno);
bnd = find(ds);
boundaries = [0 bnd' length(seqno)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
for s = 1:max(seqno)
   seq_size = lenp(s);
   lb = find(endbulges{s});
   eb_size = length(lb);
   eb_begin = lb(1);
   eb_end = lb(eb_size);

   l = boundaries(s)+1 : boundaries(s+1);

   maxs = max(svm_score(l));
   score(s) = maxs;
   m = find(svm_score(l) >= maxs - (1-Thresh)*abs(maxs));
   loopdist = x(l(m),2) * (0.5* (seq_size - eb_size));
   minlpdst = min(loopdist);
   lminloopdist = find(loopdist == minlpdst);
   m = m(lminloopdist);
   if length(m > 1)
      mscore = svm_score(l(m));
      [mxs,i] = max(mscore);
      m = m(i);
   end

   side = x(l(m),1);
   loopdist = x(l(m),2) * (0.5* (seq_size - eb_size));
   pos(s) = (1+side)/2*(eb_end + loopdist) + (1-side)/2*(eb_begin - loopdist);
end
pos = round(pos);
returnfunction [pos, score] = svm_position53(x5,svm_score5, seqno5, x3, svm_score3, seqno3, endbulges, lenp);
%[pos, score] = svm_position53(x5,svm_score5, seqno5, x3, svm_score3, seqno3, endbulges, lenp);
% postprocess svm outputs (for error analysis)
```

```
global Maxpos
method = 'bestn';
param = 1;
Maxpos = 10; % maximal number of positions returned
alpha5 = 0.6; alpha3 = 0.40; alpha_dlen = 0.4; % relative weights of 5 and 3 predictions
disp(['alpha5  alpha3  alpha_dlen'  num2str(alpha5) ' ' ...
     num2str(alpha3) ' ' num2str(alpha_dlen)]);
if size(x5,1) ~= size(svm_score5,1)
   error('x5  and svm_score5 not compatible');
end
if size(x3,1) ~= size(svm_score3,1)
   error('x3  and svm_score3 not compatible');
end
if size(x5,1) ~= size(seqno5,1)
   error('x5  and seqno5 not compatible');
end
if size(x3,1) ~= size(seqno3,1)
   error('x3  and seqno3 not compatible');
end
if max(seqno5) ~= max(seqno3)
   error('seqno5 and seqno3 not compatible');
end
if max(seqno5) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
fid = fopen('d:\rosetta\svm_light_utils1\dicer_length.out','r');
dlen = str2num(fgetl(fid));
pdlen = str2num(fgetl(fid));
fclose(fid);
dlenmin = min(dlen);
dlenmax = max(dlen);
scdlen = log(pdlen); % scale to score - heuristic!!!
scdlen = scdlen + mean(scdlen);
nseq = max(seqno5);
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds5 = diff(seqno5);
bnd5 = find(ds5);
boundaries5 = [0 bnd5' length(seqno5)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
ds3 = diff(seqno3);
bnd3 = find(ds3);
boundaries3 = [0 bnd3' length(seqno3)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
if strcmp(method, 'bestn')
   pos = zeros(nseq,2*param);
   score = zeros(nseq,param);
elseif strcmp(method, 'best plus other side')
    pos = zeros(nseq,2*2);
```

```
    score = zeros(nseq,2);
else
    error('not supported yest');
end
for s = 1:max(seqno5)
    l5 = boundaries5(s)+1   : boundaries5(s+1);
    l3 = boundaries3(s)+1 : boundaries3(s+1);

    score5 = svm_score5(l5);
    score3 = svm_score3(l3);

    seq_size = lenp(s);
    lb = find(endbulges{s});
    eb_size = length(lb);
    eb_begin = lb(1);
    eb_end = lb(eb_size);

    side5 = x5(l5,1);
    loopdist5 = x5(l5,2) * (0.5* (seq_size - eb_size));
    pos5 = (1+side5)/2.*(eb_end + loopdist5) + (1-side5)/2.*(eb_begin - loopdist5);
    pos5 = round(pos5);

    side3 = x3(l3,1);
    loopdist3 = x3(l3,2) * (0.5* (seq_size - eb_size));
    pos3 = (1+side3)/2.*(eb_end + loopdist3) + (1-side3)/2.*(eb_begin - loopdist3);
    pos3 = round(pos3);
    % initialize.  pos53(:,1) contains 5' position , pos53(:,1) contains 3' position
    pos53 = zeros(length(l5)*size(pdlen,2),2);
    score53 = zeros(length(l5)*size(pdlen,2),1);
    count = 0;
    for i = 1:length(pos5);
        pos5i = pos5(i);
        J = find(pos3 >= pos5i + dlenmin -1 & pos3 <= pos5i + dlenmax -1 & side3 == side5(i));
        for j = 1:length(J);
            count = count+1;
            pos53(count,:) = [pos5i , pos3(J(j))];
            ind =  pos3(J(j))-pos5i -dlenmin +2;
            if ind < 1 | ind > size(scdlen,2)
                disp('error')
            end

            score53(count) = alpha5*score5(i) + alpha3*score3(J(j)) + alpha_dlen*scdlen(ind);
            %score53(count) = alpha5*tanh(score5(i)) + alpha3*tanh(score3(J(j))) + alpha_dlen*scdlen(ind);
            %score53(count) = max(score5(i),score3(J(j))) + alpha_dlen*scdlen(ind);
        end
    end
    % now pick the desired positions for each sequence ,
    % e.g. 'best' , 'best plus other side', 'above thresh' 'percentile'.
    l = find(pos53(:,1) == 0);
    pos53(l,:) = [];
```

```
    score53(l) = [];
    if isempty(score53)
        error('empty score53');
    end

    [poss, scores] = choose_pos_score(pos53,score53, eb_begin, method, param);
    pos(s,:) = poss;
    score(s,:) = scores;
    if mod(s,1000) == 0
        disp([num2str(s)])
    end
end
return
function [pos, score] = choose_pos_score(pos53,score53, eb_begin, method, param)
% auxilary function
if strcmp(method, 'bestn')
    nbest = param;
    [s,I] = sort(-score53);
    pos(1:nbest,:) = pos53(I(1:nbest),:);
    score(1:nbest) = score53(I(1:nbest));
    pos = pos';
    pos = pos(:);
    pos = pos';
    score = score';
elseif strcmp(method, 'best plus other side')
    pos = zeros(2,2);
    score = zeros(1,2);

    [mx,i] = max(score53);
    pos(1,:) = pos53(i,:);
    score(1) = score53(i);

    Os = find( (pos53(:,1)-eb_begin) * (pos(1,1) -eb_begin)  < 0); %Other side
    if ~isempty(Os)
        [mx,i] = max(score53(Os));
        pos(2,:) = pos53(Os(i),:);
        score(2) = score53(Os(i));
    else
        % this may hapen when the sequence on other side was too short.
        pos(2,:) = NaN;
        score(2) = NaN;
    end
elseif strcmp(method, 'percentile')
    perc = params;
    if perc < 1
        perc = perc*100;
    end
    xp = prctile(score53, perc);
    I = find(score53 >= xp);
    [s,J] = sort(-score53(I));
```

```
   J = I(J);
   score = score53(I);
   pos = pos53(I,:);
else
   error('method not implemented');
end
returnfunction [pos, score] = svm_position53h(x5,svm_score5, seqno5, x3, svm_score3, seqno3, endbulges, lenp);
%[pos, score] = svm_position53h(x5,svm_score5, seqno5, x3, svm_score3, seqno3, endbulges, lenp);
% postprocess svm outputs (for error analysis)
% hard limiter on results of classifier on 3'
global Maxpos
method = 'bestn';
param = 1;
Maxpos = 10; % maximal number of positions returned
alpha5 = 0.6; alpha3 = 0.40; alpha_dlen = 0.4; % relative weights of 5 and 3 predictions
disp(['alpha5  alpha3  alpha_dlen'  num2str(alpha5) ' ' ...
     num2str(alpha3) ' ' num2str(alpha_dlen)]);
if size(x5,1) ~= size(svm_score5,1)
   error('x5  and svm_score5 not compatible');
end
if size(x3,1) ~= size(svm_score3,1)
   error('x3  and svm_score3 not compatible');
end
if size(x5,1) ~= size(seqno5,1)
   error('x5  and seqno5 not compatible');
end
if size(x3,1) ~= size(seqno3,1)
   error('x3  and seqno3 not compatible');
end
if max(seqno5) ~= max(seqno3)
   error('seqno5 and seqno3 not compatible');
end
if max(seqno5) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
fid = fopen('d:\rosetta\svm_light_utils1\dicer_length.out','r');
dlen = str2num(fgetl(fid));
pdlen = str2num(fgetl(fid));
fclose(fid);
dlenmin = min(dlen);
dlenmax = max(dlen);
scdlen = log(pdlen); % scale to score - heuristic!!!
scdlen = scdlen + mean(scdlen);
nseq = max(seqno5);
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds5 = diff(seqno5);
```

```
bnd5 = find(ds5);
boundaries5 = [0 bnd5' length(seqno5)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
ds3 = diff(seqno3);
bnd3 = find(ds3);
boundaries3 = [0 bnd3' length(seqno3)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
if strcmp(method, 'bestn')
   pos = zeros(nseq,param);
   score = zeros(nseq,param);
elseif strcmp(method, 'best plus other side')
    pos = zeros(nseq,2);
   score = zeros(nseq,2);
else
   error('not supported yest');
end
for s = 1:max(seqno5)
   l5 = boundaries5(s)+1   : boundaries5(s+1);
   l3 = boundaries3(s)+1 : boundaries3(s+1);

   score5 = svm_score5(l5);
   score3 = svm_score3(l3);

   seq_size = lenp(s);
   lb = find(endbulges{s});
   eb_size = length(lb);
   eb_begin = lb(1);
   eb_end = lb(eb_size);

   side5 = x5(l5,1);
   loopdist5 = x5(l5,2) * (0.5* (seq_size - eb_size));
   pos5 = (1+side5)/2.*(eb_end + loopdist5) + (1-side5)/2.*(eb_begin - loopdist5);
   pos5 = round(pos5);

   if max(score3 < -0.1)
     [maxs,m] = max(svm_score5);
     score(s) = maxs;
     pos(s) = po5(m);
   else
     side3 = x3(l3,1);
     loopdist3 = x3(l3,2) * (0.5* (seq_size - eb_size));
     pos3 = (1+side3)/2.*(eb_end + loopdist3) + (1-side3)/2.*(eb_begin - loopdist3);
     pos3 = round(pos3);
   % initialize.  pos53(:,1) contains 5' position , pos53(:,1) contains 3' position
    pos53 = zeros(length(l5)*size(pdlen,2),1);
    score53 = zeros(length(l5)*size(pdlen,2),1);
    count = 0;
    for i = 1:length(pos5);
       pos5i = pos5(i);
       J = find(pos3 >= pos5i + dlenmin -1 & pos3 <= pos5i + dlenmax -1 & side3 == side5(i));
       for j = 1:length(J);
          count = count+1;
```

```
            pos53(count,:) = pos5i;
            ind =  pos3(J(j))-pos5i -dlenmin +2;
            if ind < 1 | ind > size(scdlen,2)
               disp('error')
            end

            score53(count) = alpha5*score5(i) + alpha3*score3(J(j)) + alpha_dlen*scdlen(ind);
         end
      end
      % now pick the desired positions for each sequence ,
      % e.g. 'best' , 'best plus other side', 'above thresh' 'percentile'.
      I = find(pos53(:,1) == 0);
      pos53(I) = [];
      score53(I) = [];
      if isempty(score53)
         error('empty score53');
      end

      [poss, scores] = choose_pos_score(pos53,score53, eb_begin, method, param);
      pos(s,:) = poss;
      score(s,:) = scores;
      if mod(s,1000) == 0
         disp([num2str(s)])
      end
   end
end
return
function [pos, score] = choose_pos_score(pos53,score53, eb_begin, method, param)
% auxilary function
if strcmp(method, 'bestn')
   nbest = param;
   [s,I] = sort(-score53);
   pos(1:nbest) = pos53(I(1:nbest));
   score(1:nbest) = score53(I(1:nbest));
   pos = pos';
   score = score';
elseif strcmp(method, 'best plus other side')
   pos = zeros(1,2);
   score = zeros(1,2);

   [mx,i] = max(score53);
   pos(1) = pos53(i);
   score(1) = score53(i);

   Os = find( (pos53(:,1)-eb_begin) * (pos(1,1) -eb_begin)  < 0); %Other side
   if ~isempty(Os)
      [mx,i] = max(score53(Os));
      pos(2) = pos53(Os(i));
      score(2) = score53(Os(i));
   else
```

```
        % this may hapen when the sequence on other side was too short.
        pos(2) = NaN;
        score(2) = NaN;
    end
elseif strcmp(method, 'percentile')
    perc = params;
    if perc < 1
        perc = perc*100;
    end
    xp = prctile(score53, perc);
    I = find(score53 >= xp);
    [s,J] = sort(-score53(I));
    J = I(J);
    score = score53(I);
    pos = pos53(I  );
else
    error('method not implemented');
end
returnfunction svm_predict(infile,outfile);
%svm_predict(infile,outfile);
%perform svm position prediction
svm_light_folder = 'C:/svm/bin/';
model_filename5 = [svm_light_folder 'model5-2-6-2-6-21'];
tst_filename5 = 'C:/svm/Temp/svm_tst_5.dat';
svm_out_filename5 = 'C:/svm/Temp/out5.out';
fit_filename = 'C:/svm/Score/fit_p5-2-6-2-6-21.txt';
[seqs,bulges1,bulges2,endbulges,seq_id] = read_structure(infile);

[x5, seqno5] = preprocess_and_write_data5(seqs,bulges1, ...
   bulges2,endbulges, tst_filename5);
dos([svm_light_folder 'svm_classify ' tst_filename5 ' ' model_filename5 ' ' svm_out_filename5]);
% load and postprocess
curdir=pwd;
cd 'c:/svm/temp'
load out5.out;
cd(curdir);
lenp = length_seq(seqs);
[pos5, score5] = svm_position(x5,out5, seqno5, endbulges, lenp);
% infer probabilities
[yside, yprec2] = interpolate_prob_new(score5, fit_filename);
%write to file
res = [seq_id; pos5; score5; yprec2; yside];
fid = fopen(outfile,'w');
fprintf(fid, '%d %d %g %g %g\n', res);
fclose(fid);
function svm_predict_();
%svm_predict_b(infile,outfile);
%perform svm position prediction
% version for large input files
% reads seqtot sequences at a time and classifies them
```

```
infile='c:\svm\in\draw_file.dat';
outfile='c:\svm\out\dicer_res.dat';
cd \\rosetta4\Development\gideon\svm\util
svm_light_folder = 'C:/svm/bin/';
model_filename5 = [svm_light_folder 'model5-2-6-2-6-21'];
tst_filename5 = 'C:/svm/Temp/svm_tst_5.dat';
svm_out_filename5 = 'C:/svm/Temp/out5.out';
fit_filename = 'C:/svm/Score/fit_p5-2-6-2-6-21.txt';
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
   disp('reading structure...');
 [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);
 lenp = length_seq(seqs);

  disp('preprocessing and writing...');
 [x5, seqno5] = preprocess_and_write_data5(seqs,bulges1, ...
    bulges2,endbulges, tst_filename5);
  cd c:\
  dos([svm_light_folder 'svm_classify ' tst_filename5 ' ' model_filename5 ' ' svm_out_filename5]);
  cd \\rosetta4\Development\gideon\svm\util
% load and postprocess
 fidsvm = fopen(svm_out_filename5,'r');
 out5 = fscanf(fidsvm, '%g');
   fclose(fidsvm);

  disp('postprocessing...');
 [pos5, score5] = svm_position(x5,out5, seqno5, endbulges, lenp);
% infer probabilities
 [yside, yprec2] = interpolate_prob_new(score5, fit_filename);
%write to file
%seq_id0 is added so as to sequential order of sequence numbers
 seq_id = seq_id + seq_id0;
 res = [seq_id; pos5; score5; yprec2; yside];
   fprintf(fidout, '%d %d %g %g %g ', res);
   seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
quit
function svm_predict_b(infile,outfile);
%svm_predict_b(infile,outfile);
%perform svm position prediction
% version for large input files
% reads seqtot sequences at a time and classifies them
svm_light_folder = 'C:/svm/bin/';
model_filename5 = [svm_light_folder 'model5-2-6-2-6-21'];
tst_filename5 = 'C:/svm/Temp/svm_tst_5.dat';
```

```
svm_out_filename5 = 'C:/svm/Temp/out5.out';
fit_filename = 'C:/svm/Score/fit_p5-2-6-2-6-21.txt';
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
   disp('reading structure...');
 [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);
 lenp = length_seq(seqs);

  disp('preprocessing and writing...');
 [x5, seqno5] = preprocess_and_write_data5(seqs,bulges1, ...
    bulges2,endbulges, tst_filename5);
 dos([svm_light_folder 'svm_classify ' tst_filename5 ' ' model_filename5 ' ' svm_out_filename5]);
% load and postprocess
 fidsvm = fopen(svm_out_filename5,'r');
 out5 = fscanf(fidsvm, '%g');
   fclose(fidsvm);

  disp('postprocessing...');
 [pos5, score5] = svm_position(x5,out5, seqno5, endbulges, lenp);
% infer probabilities
 [yside, yprec2] = interpolate_prob_new(score5, fit_filename);
%write to file
%seq_id0 is added so as to sequential order of sequence numbers
 seq_id = seq_id + seq_id0;
 res = [seq_id; pos5; score5; yprec2; yside];
   fprintf(fidout, '%d %d %g %g %g\n', res);
   seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
function unique_seqs(seqs,seqsd,bulges1, bulges2, endbulges, lenp, lend, pos, seq_id)
[y,l] = sort(lenp);
 bulges1 = bulges1(l);
 bulges2 = bulges2(l);
 endbulges = endbulges(l);
 lend = lend(l);
 lenp = lenp(l);
 pos = pos(l);
 seq_id = seq_id(l);
 seqs = seqs(l);
  seqsd = seqsd(l);

   count = 1;
   lc(count) = 1;
   for i = 2:length(seqs)
      if lenp(i) == lenp(i-1)
         if any(seqs{i} ~= seqs{i-1})
```

```
        count = count+1;
        lc(count) = i;
        end
      else
        count = count+1;
        lc(count) = i;
   end
   end

   keyboard
function write_examples(xi,yi, fid);
% %low level function
% write examples in format compatible with svm light
% xi,yi are the example vector + targets
% in testing mode the yi's are set to 0
for j = 1:size(xi,1)
    fprintf(fid,'%d',yi(j));
    I = find(xi(j,:));
    xprint = [I;xi(j,I)];
    fprintf(fid,' %d:%g',xprint);
    fprintf(fid,'\n');
endfunction write_examples_simple(xi,yi, fid);
% %low level function
% write examples in format compatible with svm light
% xi,yi are the example vector + targets
% in testing mode the yi's are set to 0
for j = 1:size(xi,1)
    fprintf(fid,'%d',yi(j));
    fprintf(fid,' %g',xi);
    fprintf(fid,'\n');
endkkk = 1:6000;
[x12_5, seqno_5] =
preprocess_and_write_data5(seqs(kk),bulges1(kk),bulges2(kk),endbulges(kk),'g:\research\rosetta\svm_light_utils1\sv
m_preprocessed\svm5_kk1.dat');
[x12_3, seqno_3] =
preprocess_and_write_data5(seqs(kk),bulges1(kk),bulges2(kk),endbulges(kk),'g:\research\rosetta\svm_light_utils1\sv
m_preprocessed\svm3_kk1.dat');
% use svm light to classify both each ewth its own model
load output5_kk1.out
load output3_kk1.out
[pos53kk, score53kk] = svm_position53(x12_5,output5_kk1, seqno_5, x12_3, output3_kk1, seqno_3, endbulges(kk),
lenp(kk));
%write final results to file
res = [seq_id(kk); pos53kk(:,1)'; pos53kk(:,2)'; score53kk'];
fid = fopen('res53_33156.out','a');
fprintf(fid, '%d %d %d %g\n', res);
fclose(fid);
```